10同步、调度

2019年2月14日 16:38

-
- 进程同步

1. 紧密耦合系统可直接通过共享存储时限同步

一. 集中式与分布式同步方式

1. 同步实体Synchronizing Entity：实现同步的实体，如硬件锁、信号量、进程
    1) 中心同步实体：满足以下条件的同步实体：
        (1) 有唯一的名字，被所有需同步的进程知道
        (2) 任何时刻都可被这些进程访问
    2) 中心同步实体的容错技术：失效时，系统会立即选一个新的投入运行
2. 集中式同步机构：基于中心同步机构的同步机构
    1) 对同一处理机的同步机制：硬件锁、信号量等
    2) 对不同处理机的：自旋锁、RCU锁、时间邮戳、事件计数、中心进程等
3. 集中式与分布式同步算法
    1) 集中式同步算法
        (1) 多进程访问共享资源或通信时，仅由中心控制结点判定出一个进程
        (2) 判定所需信息都集中在中心控制结点
        (3) 适用于单处理机系统和共享存储器的多处理机系统；分布式系统适合分布式同步算法；松散耦合式则两种都可能采用
        (4) 缺点：
            i. 可靠性差，结点故障易形成灾难性影响
            ii. 易形成瓶颈，管理大量资源的结点影响到系统的响应速度
    2) 分布式同步算法
        (1) 所有结点有相同信息
        (2) 所有结点仅基于本地信息做出判定
        (3) 为做出判定，所有结点担负相同职责，付出相同工作量
        (4) 一个结点发生故障通常不导致整个系统的崩溃
4. 中心进程方式
    1) 中心进程/协调进程：保存所有用户的存取权限、冲突图conflict graph等
    2) 访问共享资源前需要先向中心进程发送一条请求信息，中心进程查冲突图确认不会引起死锁后将该请求插入请求队列，否则退回rollback
    3) 当资源空闲，中心进程向队首进程发出消息，允许使用；释放资源时也要向中心进程发出消息。进出临界区必须有请求、回答、释放消息
    4) 为提高可靠性，中心进程可以浮动

二. 自旋锁spin lock

1. 自旋锁的引入
    1) 读-改-写操作需要执行多次总线操作，若执行原语过程中，总线被其他CPU争得，可能导致该存储单元被交叉操作，破坏原语的原子性

2) 自旋锁机制可用于但不局限于对总线资源的竞争

2. 实现互斥访问总线的方法
   1) 在总线设置一个自旋锁，该锁最多只能被一个内核进程使用
   2) 获得不到自旋锁的进程会“自旋”循环测试锁的状态，直至得到
3. 自旋锁与信号量的主要差别
   1) 自旋锁可避免调用进程阻塞，因为自旋锁保护的临界区一般较短，不会产生“忙等”，信号量机制切换进程花费开销大，使用自旋锁的效率更高
   2) 若不需中断上下文、或需共享设备、或临界区较大，还是该用信号量
   3) 自旋锁保持期间不可抢占，只在内核可抢占或SMP时才真正需要；单CPU可通过关中断防止中断处理的并发，自旋锁的操作是空操作
4. 自旋锁的类型
   1) 普通自旋锁：可获得时才为0，使用自旋锁不影响当前处理机的中断状态
   2) 读写自旋锁：允许被多个读者只读访问，需要读者数计数和解锁标记
   3) 大读者自旋锁：获取读锁时只需对本地读锁加锁，开销小；获取写锁时必须锁住所有CPU的读锁，代价高
   4) 基本形式：

```
spin_lock(&lock);
/*临界区*/
spin_unlock(&lock);
```

三. 读-拷贝-修改锁
1. Read-Copy-Update锁的引入
   1) 只要有一个进程在写，多个读进程会同时被阻塞，严重影响读工作
   2) 可以通过让写进程先用读（拷贝）出一个副本，对副本修改，再写回去
2. RCU锁
   1) 读不需任何锁，写只需制作出副本，再在适当时机利用回调机制
   2) 回调callback机制：向垃圾收集器机构注册一个回调函数，负责让原数据指向新数据，并释放数据
3. 写回时机
   1) 规定读任务结束后需要提供一个信号，所有读者都发送信号时即可修改
   2) 延迟期grace period：副本修改完到等待信号之间的时间
4. RCU锁的优缺点
   1) 读者不会被阻塞，提高了读进程的运行效率并减少了CPU上下文处理开销
   2) 无需为共享数据设置同步机构：对读者来说，没什么同步开销，也不怕死锁；不过写者需要复制文件、延迟释放、同步其他写操作
   3) 不适用于写操作多于读操作的情况，对读的性能提高可能不足以弥补写的损失，此时适合读写自旋锁

四. 二进制指数补偿算法和待锁CPU等待队列机构
1. 二进制指数补偿算法：对CPU测试锁的TSL指令设置延迟执行时间，该时间每次测试后扩大到该次延迟时间的两倍，直至到达一个最大值
   1) 可明显降低总线上的数据流量，因为减少了测试次数和频率

2) 缺点是延迟时间可能导致空闲的锁不能被及时使用，造成浪费

2. 待锁CPU等待队列机构：在每个CPU的高速缓存中配一个用于测试的私有锁变量和待锁CPU清单，访问共享数据失败的CPU会被分配到锁变量，并被添加到正在使用该数据的CPU的清单末，之后第n个CPU将被添加到第n-1个CPU后面

1) 当共享数据占用者退出临界区时，检查高速缓存，释放私有锁变量

2) 每个待锁CPU都仅在自己的高速缓存中不断测试私有锁，不访问总线

3) CPU释放锁后应及时释放清单下一个CPU的锁，避免浪费空闲资源

五. 定序机构

1) 多处理机系统和分布式系统中，每个系统都有自己的物理时钟

2) 定序机构负责排序各系统中所有特定事件，保证各处理机的进程协调运行

1. 时间邮戳定序机构Timestamp Ordering Mechanism：用系统中唯一的、单一物理时钟驱动的物理时钟体系，确保各处理机时钟严格同步。其基本功能：

1) 对资源请求、通信等特殊事件加印上时间邮戳

2) 对每种特殊事件只能用唯一的时间邮戳

3) 根据时间邮戳定义所有事件的全序

2. 事件计数Events Counts同步机构

1) 定序器sequencer：用于为所有特定事件排序的整型量

2) 定序器初值为0，非减少，只能被施加ticket操作

3) 事件刚发生时会被系统分配标号V，之后ticket自动+1

4) 系统会将已服务事件的标号保留，形成事件计数栈E，其值为栈顶标号

(1) await(E,V){ //进程进入临界区前需执行await

```
if(E<V){
    i=EP;
    stop();
    i->status="block";
    i->sdata=EQ;
    insert(EQ,i); //将执行进程插入EQ队列
    scheduler();
}else continue;}
```

(2) advance(E){ //进程退出临界区后需执行advance

```
++E;
if(EQ!=NIL){
    V=inspect(EQ,1);
    if(E==V)
        wakeup(EQ,1);}}
```

(3) read(E)：返回E当前值

(4) 以上三个操作在同一事件上可并发执行，但定序器必须互斥使用

六. 面包房算法

1) 是最早的分布式同步算法，通过给时间排序，再按FCFS处理

1. 系统由n个结点组成，每个结点仅有一个进程，仅负责处理一种临界资源

2. 每个进程保持一个队列，用来记录按事件时序排序的收到的消息和产生的消息
3. 消息分为：请求消息、应答消息、撤销消息
4. 进程Pi发送的请求消息形如request(Ti,i)。Ti=Ci是发送时逻辑时钟值，i是内容
5. 面包房算法描述：
    1) Pi请求资源时，把request(Ti,i)排在自己的请求队列中，并发给其他进程
    2) Pj收到消息后，放入自己的请求队列中，再发送回答reply(Tj,j)
    3) 满足以下条件时，允许Pi进入临界区/访问该资源
        (1) Pi的该请求消息处于请求队列最前
        (2) Pi收到所有其他进程发来的回答消息，时间戳均晚于Ti
    4) 释放资源时要从队列中撤销该请求，再发送打上时间戳的release消息给其他进程，收到该消息的进程也撤销队列中的该请求

七. 令牌环算法

1. 也是分布式同步算法，会将所有进程组成一个逻辑环Logical Ring
2. 令牌Token：特定格式的报文，在逻辑环中被循环传递，获得令牌的进程有权进入临界区访问共享资源，由于只能被一个进程持有，实现了互斥
3. 令牌初始化后随机赋予逻辑环任一进程，以点对点的形式按固定方向和顺序依次逐个传到下一个进程，不需访问共享资源的话就不保持令牌，直接传递
4. 缺点是令牌丢失或被破坏时难以检测和判断，如通信链路、进程故障等，需要及时屏蔽故障，重构逻辑环，重颁令牌等

-
- 多处理机系统的进程调度

一. 评价调度性能的若干因素

1. 任务流时间：完成任务所需时间
2. 调度流时间：系统中所有处理机的任务流时间综合
3. 平均流：调度流时间除以任务数
    1) 平均流小反应了资源利用率高，任务的机时费用低，完成任务时间充裕
    2) 最少平均流时间是系统吞吐率的间接度量参数
4. 处理机利用率：任务流之和除以最大有效时间单位
5. 加速比：各处理机忙时间之和除以并行工作时间（开始运行到全结束的时间）
    1) 加速比用于度量多处理机系统的加速程度
6. 吞吐率：单位时间内完成的任务数，一般用任务流最小完成时间来度量
    1) 与调度算法复杂性有密切关系
    2) 求解最优调度是Nondeterministic Polynomial完全性问题
    3) 难以求得最坏情况下的最优调度，一般只考虑典型情况的合适调度

二. 进程分配方式

1. 对称多处理机系统中的进程分配方式
    1) 静态分配Static Assignment方式：从开始执行到完成都在同一处理器上
        (1) 为每一处理器设置一专用的就绪队列，之后同单处理机系统
        (2) 调度开销小；各处理器忙闲不均
    2) 动态分配Dynamic Assignment方式：每次被调度都随机分到任一处理器

(1) 系统中仅设置一个公共就绪队列

(2) 消除了忙闲不均现象；适用于紧密耦合系统，但对松散耦合系统需要传递前一次运行时的处理器上下文，增加了调度开销

2. 非对称MPS中的进程分配方式：由主机为发出空闲信号的从机分配进程。为防止主机故障导致的系统瘫痪和主机太忙形成的系统瓶颈，可设置多台主机

三. 进程/线程调度方式

1. 自调度Self-Scheduling方式

1) 自调度机制：在系统中设置一个公共就绪队列，空闲处理机自行调度。由于多处理机使等待速度减小，FCFS的效率高于优先权调度

2) 优点：可以简单地沿用单处理机系统调度算法，不容易忙闲不均

3) 缺点：互斥访问队列易形成瓶颈；阻塞后的重运行需重新建立上下文，低效；合作性线程难以同时运行，导致切换频繁

2. 成组调度Gang Scheduling方式

(1) 由Leutenegger提出，将同一进程的线程成组分配

(2) 合作线程的并行执行能减少阻塞，改善系统性能

(3) 每次调度可解决一组线程，减少调度频率和开销，性能优于自调度

1) 面向所有应用程序平均分配处理器时间：M个进程各被分配约1/M的时间

2) 面向所有线程平均分配处理机时间：N个线程各被分配约1/N的时间

3. 专用处理机分配Dedicated Processor Assignment方式

1) 1989由Tucker提出，在运行期间，专门分配一组处理机，直至完成

2) 易造成处理机空闲的严重资源浪费，不适用于处理机少的环境

3) 但对数十个处理机高度并行的系统来说，各处理机的投资费用只占很小一部分，每个线程专用一台处理机可有效避免切换，加速运行

(1) 如16个处理机的系统运行矩阵相乘和傅立叶变换（FFT）两个程序

(2)

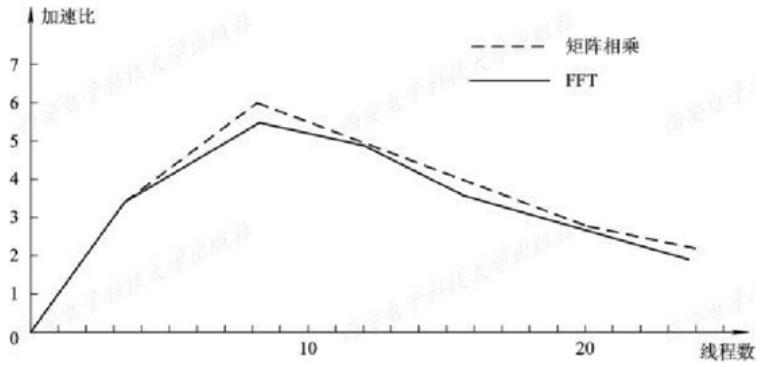

图10-8 线程数对加速比的影响

(3) 如图，线程数总和接近处理机数时加速比最高，不能保证线程数<=处理机数时，线程数越多，加速比越低。类似物理块数少于工作集数时会频繁导致缺页

4. 动态调度

1) 允许进程在执行期间动态改变线程数，由系统和程序共同进行调度决策

2) 系统负责分配处理机给作业，作业自行分配处理机执行部分任务

3) 系统分配处理机的原则

(1) 空闲则分配：有作业提出处理机请求时，将空闲处理机分配给作业
(2) 新作业绝对优先：优先分配给无任何处理机的新到作业，必要时收回分配给旧作业的处理机
(3) 保持等待：指系统任何分配都不能满足作业时需保持等待新处理机（或由作业自己取消处理及请求）
(4) 释放即分配：释放处理机后立刻分配给新作业，再按FCFS

4) 优于前两种调度，但开销过大

四. 死锁

1. 死锁的类型
   1) 资源死锁：竞争可重用资源（打印机、存储器等）或推进顺序不当。如集中式系统中互相等对方发送消息
   2) 通信死锁：主要是分布式系统不同结点中的进程因报文而竞争缓冲区
2. 死锁的检测和解除
   1) 集中式检测：各处理机都有一进程资源图描述进程及其占有资源，再由中心处理机配置全系统的进程资源图，并设置检测进程及时中止环路的进程
      (1) 检测方式：
         i. 当资源图中加入或删除弧时将相应变动消息发给检测进程
         ii. 由进程自行将弧的变动信息周期性地发送给检测进程
         iii. 检测进程主动请求更新信息
      (2) 缺点是发出消息与执行命令时序不一定相同。可以在检测到环路后重新向进程发出请求，若收到了否认的回答就说明未死锁
   2) 分布式检测：系统中竞争资源的进程相互协作，自行检测
      (1) 在每个结点中都设置一个检测进程，每个消息上都附加逻辑时钟，并依次对请求和释放资源的消息排队
      (2) 请求某资源前需要给所有其他进程发送请求信息，获得全部响应后才能把请求资源的消息发给管理进程，分配情况也要通知所有进程
      (3) 可见，分布式环境死锁检测的通信开销较大，一般是靠死锁预防
         i.
         ii.
         iii.
         iv.
         v.
         vi.
         vii. ---------我是底线-------